10/719,443 PN-892

(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 1 349 077 A2

(12) EUROPEAN PATENT APPLICATION

(43) Date of publication:
01.10.2003 Bulletin 2003/40

(51) Int Cl.[7]: **G06F 12/02**, G06F 11/34

(21) Application number: **03250706.3**

(22) Date of filing: **04.02.2003**

(84) Designated Contracting States:
AT BE BG CH CY CZ DE DK EE ES FI FR GB GR HU IE IT LI LU MC NL PT SE SI SK TR
Designated Extension States:
AL LT LV MK RO

(30) Priority: **21.03.2002 US 104751**

(71) Applicant: **Hewlett-Packard Company**
Palo Alto, CA 94304 (US)

(72) Inventors:
- **Coha, Joseph A.**
  San Jose. CA 95129 (US)
- **Karkare, Ashish**
  San Jose, CA 95131 (US)
- **O'Konski, Timothy C.**
  Palo Alto, CA 94301 (US)

(74) Representative: **Jehan, Robert et al**
Williams Powell,
Morley House,
26-30 Holborn Viaduct
London EC1A 2BP (GB)

(54) Optimization of memory usage based on garbage collection simulation

(57) A method for optimization of memory usage for a computer application. Memory usage data (720) is received (215) wherein the memory usage data (720) comprises timing information. A graphical representation (740) of the memory usage data (720) is generated (225). At least one heap parameter (725) is received (230). A memory usage simulation is performed (245) based on the memory usage data (720) and the heap parameter (725).

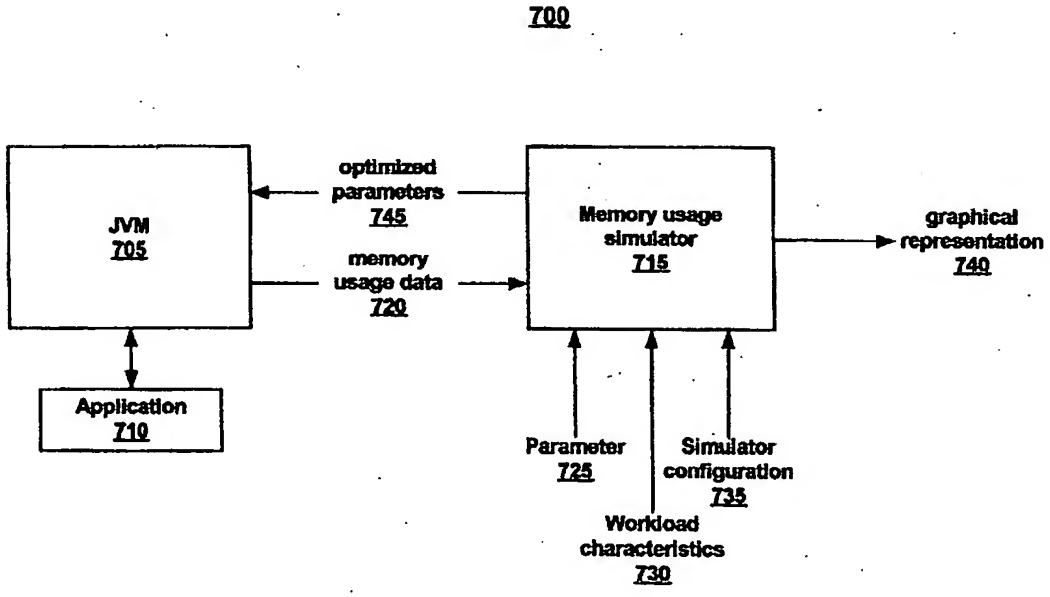


FIG. 7

EP 1 349 077 A2

Description

[0001] The present invention relates to a method of and apparatus for optimising memory usage.

[0002] Many modern programming languages provide facilities for dynamic allocation of memory for objects or other data structure instances created by applications at run-time. Some languages such as C and C++ require explicit user action for reclaiming such dynamically allocated memory, while other languages such as Java and C# provide support for memory management that enables automatic reclamation of dynamically allocated memory. For example, the Java programming language allocates objects within the Java Virtual Machine (JVM) heap. When an object is no longer required, it is not explicitly released or removed from the heap. In order to remove unused objects, Java provides a built in capability for the collection of unreferenced dynamically allocated objects, herein referred to as "garbage." The automatic memory management capability is commonly referred to as "garbage collection".

[0003] Garbage collection provides the ability to release and reclaim memory when it is no longer required, thus providing a paradigm that enables the creation of more efficient applications. However, garbage collection itself adds a performance overhead to many Java applications. In particular, Java enterprise applications that utilize significant system memory resources can be performance or capacity impaired based on the garbage collection activity that they produce as the workload they are processing increases. This leads to slow and sometimes unreliable operation of Java enterprise applications.

[0004] Except for the situations where garbage collection overhead is a direct result of software defects, non-optimal heap memory usage and incorrect policy choices for the garbage collector are the primary causes of increased garbage collection overhead. The process of configuring the heap for optimal memory usage and fine-tuning the garbage collector policies is referred to as "garbage collector tuning."

[0005] Garbage collector tuning can yield a very large performance payoff (e.g., 100 percent, or even substantially higher) on certain enterprise applications by optimizing the heap memory usage and reducing the total time spent in garbage collection. This enables one to achieve predictable and optimal performance of applications, written in Java or similar programming languages.

[0006] Current garbage collection tools are integrated as part of Java profiling or monitoring solutions. These tools typically rely on measuring heap capacity using either published Java interfaces, or rely on the garbage collection trace option available in an implementation of the JVM. While these tools provide a user with some guidance for optimizing garbage collection, these tools have a number of drawbacks. Most significantly, these tools do not provide any guidance on the actual optimal garbage collection parameters, and provide no relief from the expensive process described below.

[0007] Figure 1 is a flowchart diagram illustrating a process 100 for garbage collection tuning in accordance with the prior art. At step 105, the memory allocation behavior trace used to configure the garbage collection is specified to the interpreter. At step 110, an application is executed for the programming language which the application was written in. At step 115, the trace data is analyzed. At step 120, it is determined whether or not the desired results were achieved. If so, process 100 ends. Alternatively, if desired results were not achieved, as shown at step 125, the heap parameters are adjusted based on the analysis and process 100 returns to step 105.

[0008] Typically, garbage collection is subject to user-entered or default parameters. In order to determine which parameters to insert, a user must run the application and analyze the results of the Java heap behavior. For example, a user must perform a run of the application to allow for collection of heap behavior data, and analyze the results. While the overall garbage collection time may be small, garbage collections may take a substantial amount of the overall execution time for large applications, such as enterprise applications. For example, the garbage collection on a large application dominate the overall execution time. Upon completing a run of the application, the heap behavior data is analyzed and the new parameters are specified by the user based on this analysis. This process is repeated until the desirable parameters are entered.

[0009] Furthermore, in order to optimize garbage collection, a user must determine new parameters whenever the application is changed or whenever the platform is changed. The time spent running the application, collecting the heap behavior data, and analyzing the results can be sizable, resulting in substantial costs to those running the applications.

[0010] Additional problems arise when the application being analyzed is being diagnosed off-site. It is typically too cumbersome to install an application, particularly on a third party computer system. Furthermore, the owner of the application may not want to install the application on a third party computer system due to privacy concerns. As such, a third party must tell the user which parameters to enter, then wait until the application is run, and the heap behaviour data is collected. This consumes the computing resources of the application owner, and may impose extra costs beyond those associated directly with running the application (e.g., network downtime and lowered performance of other applications).

[0011] The present invention seeks to provide improved memory usage.

[0012] According to an aspect of the present invention there is provided a method of optimising memory usage as specified in claim 1.

[0013] According to another aspect of the present in-

vention there is provided a computer readable medium having computer-readable program code embodied therein for causing a computer system to perform a method for simulating collection of unreferenced objects, said method comprising:

a) accessing unreferenced object collection data comprising timing data;
b) illustrating a viewable depiction of said unreferenced objection collection data;
c) accessing at least one memory criterion;
d) executing an unreferenced objection collection simulation based on said unreferenced objection collection data and said memory criterion; and
e) illustrating a viewable depiction of said garbage collection simulation.

[0014] According to another aspect of the present invention there is provided a computer system comprising:

a bus;
a computer-readable memory coupled to said bus; and
a processor coupled to said bus, said processor for performing a method of garbage collection simulation, said method comprising:

a) receiving garbage collection data comprising timing data;
b) rendering a visual portrayal of said garbage collection data;
c) receiving at least one heap input;
d) executing a garbage collection simulation based on said garbage collection data and said heap input; and
e) rendering a visual portrayal of said garbage collection simulation.

[0015] According to another aspect of the present invention there is provided a method of simulating memory usage comprising:

a) receiving memory usage data;
b) determining a simulation model based on said memory usage data;
c) determining heap configuration and heap policies based on said memory usage data;
d) receiving at least one heap parameter;
e) determining simulation characteristics for at least one measurement interval of said memory usage data; and
f) simulating an application run based on said simulation characteristics.

[0016] In the method for optimization of memory usage for a computer application presented, memory usage data is received wherein the memory usage data comprises timing information. A graphical representation of the memory usage data is generated. At least one heap parameter is received. A memory usage simulation is performed based on the memory usage data and the heap parameter.

[0017] Embodiments of the present invention are described below, by way of example only, with reference to the drawings, in which:

FIGURE 1 is a flowchart diagram illustrating a process for garbage collection tuning in accordance with the prior art.

FIGURES 2A and 2B are a flowchart diagram illustrating a process for selecting a parameter for garbage collection tuning in accordance with an embodiment of the present invention.

FIGURE 3 is an exemplary screen shot of garbage collection duration view generated by a garbage collection simulator graphical user interface in accordance with one embodiment of the present invention.

FIGURE 4 is an exemplary screen shot of a summary view generated by a garbage collection simulator graphical user interface in accordance with one embodiment of the present invention.

FIGURE 5 is an exemplary screen shot of a heap parameter input view generated by a memory usage simulator graphical user interface in accordance with one embodiment of the present invention.

FIGURE 6 is a flowchart diagram illustrating a process for simulating a garbage collection in accordance with one embodiment of the present invention.

FIGURE 7 is a data flow diagram illustrating data flow between a Java Virtual Machine and a garbage collection simulator in accordance with one embodiment of the present invention.

[0018] Reference will now be made in detail to the preferred embodiments of the invention, examples of which are illustrated in the accompanying drawings. The disclosure is intended to cover alternatives, modifications and equivalents, which may be included within the scope of the invention as defined by the appended claims. Furthermore, in the following detailed description, numerous specific details are set forth in order to provide a thorough understanding of the teachings herein. However, it will be apparent to one skilled in the art that they may be practised without these specific details.

[0019] Figures 2A and 2B are a flowchart diagram illustrating a process 200 for simulating programming language memory usage in accordance with an embodiment of the present invention. In one embodiment,

process 200 is carried out by processors and electrical components under the control of computer readable and computer executable instructions. The computer readable and computer executable instructions reside, for example, in data storage features such as a computer usable volatile memory and/or computer usable non-volatile memory. However, the computer readable and computer executable instructions may reside in any type of computer readable medium. Although specific steps are disclosed in process 200, such steps are exemplary. That is, the embodiments of the present invention are well suited to performing various other steps or variations of the steps recited in Figures 2A and 2B.

[0020] At step 205 of Figure 2A, the memory allocation behavior trace used to configure the garbage collection is specified to the interpreter. In one embodiment, the interpreter is a Java Virtual Machine (JVM). The memory allocation behavior trace is a mechanism that captures information that is necessary and sufficient for conducting an offline simulation and that can be collected with minimal intrusion on the executing application. In one embodiment, the memory allocation behavior trace is the -Xverbosegc option on HP-UX. In one embodiment, the memory allocation behavior trace data comprises information regarding the collection of garbage, and is also referred to herein as memory usage data. For clarification, memory usage data is also referred to herein as garbage collection data, memory allocation behavior trace, and detailed garbage collection trace data. Generally, the memory usage data is a file in tabular form comprising detailed information regarding events and operations occurring during a garbage collection. In one embodiment, the memory usage data includes, but is not limited to:

- why the garbage collection was performed;
- how long the garbage collection took;
- the time the garbage collection occurred;
- how much memory was collected by this specific garbage collection (e.g., how efficient this particular garbage collection was);
- detailed information about heap capacity;
- what portion of each of the individual segments was used before the garbage collection;
- what portion of each of the individual segments was used after the garbage collection; and
- what is the approximate age of objects deemed to be "old".

[0021] In one embodiment, where the application is Java-based, the memory usage data is produced by selecting the -Xverbosegc option in the JVM garbage collector.

[0022] At step 210, an application is executed for the programming language which the application was written in. In one embodiment, the application is an enterprise application. In one embodiment, the application is a Java-based application, written in a Java-based programming language. In the present embodiment, the interpreter is a JVM. For purposes of the present application, the interpreter is referred to as a JVM for Java-based programming languages, however, it should be appreciated that this embodiment is intended for use with any programming language that utilizes collection of unreferenced dynamically allocated objects (e.g., garbage) to free unused memory.

[0023] At step 215, the garbage collection data (e.g., memory allocation behavior trace data, memory usage data, unreferenced object collection data) is received at a garbage collection simulator. In one embodiment, where the garbage collection simulator resides in the JVM, the memory usage data is accessed directly. In another embodiment, the memory usage data is transmitted to the garbage collection simulator as an email attachment. In another embodiment, the memory usage data is transmitted as a file on a floppy disk. It should be appreciated that the memory usage data is generally a small file, and is easy to transfer from computer system to computer system.

[0024] At step 220, the garbage collection simulator is accessed. The garbage collection simulator comprises a graphical user interface that provides a user-friendly mechanism for analyzing and adjusting heap parameters, and for viewing the results of a garbage collection simulation. In one embodiment, the garbage collection simulator resides on the same computer system as the JVM. In another embodiment, the garbage collector simulator is accessed over a network connection.

[0025] At step 225, a graphical representation of the memory usage data is generated. As described above, the memory usage data is in tabular form, and is amenable to having a plurality of data plots generated therefrom. In one embodiment, a plurality of data plots are generated to illustrate different aspects of the garbage collection.

[0026] In particular, the timing information as captured in the memory usage data allows for plotting various information versus the overall time of the application's execution. Figure 3 is an exemplary screen shot 300 of a garbage collection duration data plot 305 generated by a garbage collection simulator graphical user interface in accordance with one embodiment of the present invention

[0027] Data plot 305 comprises an x-axis 310 comprising the overall application run time in seconds and a y-axis 315 comprising the run time of a specific garbage collection routine in seconds. Furthermore, data plot 305 illustrates the duration of four types of garbage collection routines: scavenge, system.gc, old full, and old too full. Specifically, data plot 305 illustrates when a garbage collection occurred, what type of garbage collection occurred, and how long it took to run. Overall timing information, such as the average scavenge garbage collection duration and the average full garbage collection duration are shown in timing information 320.

[0028] With reference to step 225 of Figure 2A, a wide

variety of data plots or graphical representations can be generated using the memory usage data. In one embodiment, additional data plots include, but are not limited to: heap usage versus time; cumulative bytes freed versus time; and creation rate. Embodiments of the present invention are directed to permitting user-defined graphical representations of data. A user desiring specific garbage collection information may generate a data plot based on requested inputs.

[0029] In one embodiment, the memory usage data is presented in a text format illustrating a summary of garbage collection activity during the application run time. Figure 4 is an exemplary screen shot 400 of summary view 405 generated by a garbage collection simulator graphical user interface in accordance with one embodiment of the present invention. Summary view 405 illustrates detailed information concerning the heap capacity of the JVM, the garbage collection activity, and overall statistics. Timing information 410 comprises information detailing how much time was spent in garbage collection. Pie chart 415 is another graphical representation of the fraction of overall run time spent in garbage collection.

[0030] With reference to step 225 of Figure 2A, a user is presented with a number of graphical representations of garbage collection information. Using these graphical representations, a user can analyze the garbage collection activities of an application. Based on the information presented in the graphical representations, a user may desire to change various input parameters in order to optimize the performance of the application.

[0031] At step 230 of Figure 2B, at least one heap parameter (e.g., a parameter, a memory criterion, or a heap input) is received at the garbage collection simulator. In one embodiment, a user-entered heap parameter is received at the garbage collection simulator. It should be appreciated that a heap parameter can be computer generated, and that this embodiment is not intended to be limited to user-generated input for heap parameters.

[0032] Figure 5 is an exemplary screen shot 500 of a heap parameter input view 505 generated by a garbage collection simulator graphical user interface in accordance with one embodiment of the present invention. Heap parameter input view allows for a user to configure the garbage collection of an application by altering various heap parameters. In one embodiment, there are four heap parameters: total heap size 510, new generation size 515, survivor ratio 520, and calls to system.gc indicator 525.

[0033] In one embodiment, heap parameter input view 505 presents the heap parameters as they exist in the application as run. A user then changes the values of the heap parameters relative to the original value. For example, where the JVM has a total heap size 510 of 64 MB, the heap parameter input view illustrates the value 64 MB associated with total heap size 510. A user may then change the value of total heap size 510, having been made aware of the original value.

[0034] In one embodiment, the heap parameter values for total heap size 510, new generation size 515 and survivor ratio 520 can be changed by directly entering text into a text box (e.g., text box 512a-c). In one embodiment, the heap parameter values for total heap size 510, new generation size 515 and survivor ratio 520 can be changed by moving a slider (e.g., slider 514a-c). In one embodiment the value depicted in text box 512 is updated as slider 514 is moved. In another embodiment, the value depicted by slider 514 is updated as a new value is entered into text box 512.

[0035] In one embodiment, indicator 525 is a check box indicating whether application initiated requests to perform memory management are included in the application run. If the application has initiated such requests, based on the trace data, this box is initially checked. A user may check or uncheck the box depending on the nature of the application.

[0036] In one embodiment, heap parameter input view 505 comprises a JVM selector 530. In one embodiment, the JVM selector 530 is a pull-down menu comprising a plurality of JVM versions. JVM selector initially illustrates which JVM version/type was used in the application run, with the ability to produce JVM options for other JVM versions/types. A user can change the JVM version/type options that are necessary for re-running their application by selecting a different JVM version. Once a user has entered in the desired heap parameters, a simulation is executed by activating simulation activator 540.

[0037] In one embodiment, heap parameter input view 505 comprises JVM options 535. JVM options 535 is a text string comprising the values selected for total heap size 510, new generation size 515, survivor ratio 520, and calls to system.gc indicator 525. JVM options 535 can be copied directly in the JVM once the desired heap parameters have been attained.

[0038] At step 235 of Figure 2B, the workload characteristics are modified. It should be appreciated that step 235 is optional, and that the workload characteristics may remain the same. At step 240, the garbage collection simulator configuration is modified. It should be appreciated that step 240 is optional, and that the garbage collection simulator configuration may remain the same.

[0039] At step 245 of Figure 2B, a garbage collection simulation (e.g., memory usage simulation) is executed. In one embodiment, selecting a simulation activator (e.g., simulation activator 340 of Figure 3) activates the garbage collection simulation. The garbage collection simulation is based on the memory usage data and the input parameters.

[0040] Figure 6 is a flowchart diagram illustrating a process 600 for simulating a garbage collection (e.g., memory usage) in accordance with an embodiment of the present invention. At step 605, detailed garbage collection trace data is received. In one embodiment, the garbage collection trace data is acquired through the

-Xverbosegc option on HP-UX. In one embodiment, the garbage collection trace data comprises information regarding the collection of garbage, and is also referred to herein as garbage collection data.

[0041] At step 610, it is determined what simulation model should be used based on the format of the garbage collection trace data. At step 615, the original heap configuration and policies are determined based on the input garbage collection trace data. At step 620, at least one adjustment to a heap parameter is received. It should be appreciated that the adjustments may be made to the heap configuration, the work load characteristics, or the simulator parameters.

[0042] At step 625, simulation characteristics are determined for each measurement interval represented in the trace data. In one embodiment, the simulation characteristics comprise:

- determine total memory allocated;
- determine total memory that is collected;
- estimate the memory allocation rate; and
- estimate the rate at which memory becomes unreferenced.

[0043] At step 630, a new application run is simulated based on the simulation characteristics determined at step 625. In one embodiment, the garbage collection simulation data estimated for the modified heap configuration includes, but is not limited to:

- the intervals between garbage collection;
- the duration of each of the individual collections;
- the cause for each garbage collection;
- the total memory allocated between two collections;
- the total memory collected between two collections;
- the duration of each individual garbage collection; and
- an estimation of the distribution of memory usage by age.

[0044] At step 250 of Figure 2B, an analysis and graphical representation of the garbage collection simulation data is generated. In one embodiment, the garbage collection simulation data is in tabular form, and is amenable to having a plurality of data plots generated therefrom. In one embodiment, a plurality of data plots are generated to illustrate different aspects of the garbage collection simulation. In particular, garbage collection simulation data comprises timing information that allows for plotting various information versus the overall time of the application's execution.

[0045] It should be appreciated that the graphical representation generated for the garbage collection simulation data is similar to that generated for the garbage collection data at step 225 of Figure 2A. For example, exemplary screen shot 300 of Figure 3 is also an illustration of a garbage collection duration data plot 305 based on the garbage collection simulation data. Likewise, exemplary screen shot 400 of Figure 4 is also an illustration of summary view 405 based on the garbage collection simulation data.

[0046] As with the garbage collection data, a wide variety of data plots or graphical representations can be generated using the garbage collection simulation data. In one embodiment, additional data plots include, but are not limited to: heap usage versus time; cumulative bytes freed versus time; and creation rate. Embodiments of the present invention are directed to permitting user-defined graphical representations of data. A user desiring specific garbage collection information may generate a data plot based on requested inputs.

[0047] In one embodiment, a graphical representation is generated comparing the garbage collection simulation data with the garbage collection data. In another embodiment, a graphical representation is generated comparing the garbage collection simulation data with another set of garbage collection simulation data. It should be appreciated that any number of garbage collection simulation data sets can be compared. The comparison graphical representation allows a user to analyze the performance of a plurality of different sets of heap parameters for the same application, thus allowing the user to select a desirable set of heap parameters for the application.

[0048] A user is presented with a number of graphical representations of garbage collection simulation information. Using these graphical representations, a user can analyze the garbage collection activities of an application. Based on the information presented in the graphical representations, a user may desire to change various input parameters in order to optimize the performance of the application and rerun the simulation.

[0049] At step 255, it is determined whether or not to run another simulation. In one embodiment, this determination is made by a user. Provided the user decides to run another simulation, process 200 returns to step 230, where at least one new heap parameter is entered. Alternatively, provided the user does not want to run another simulation, process 200 continues on to step 250.

[0050] At step 260 of Figure 2A, the optimized heap parameters are inserted into the JVM. In one embodiment, the heap parameters are provided by the garbage collection simulator as a character string (e.g., JVM options 535 of Figure 5). In one embodiment, the character string is inserted directly into the JVM by the garbage collection simulator. In another embodiment, a user can copy and paste the character string from the garbage collection simulator directly into the JVM.

[0051] At step 265, it is determined whether or not to continue the analysis of the simulation results is. In one embodiment, this determination is made by a user. Provided the user decides to continue the analysis, process 200 returns to step 210. Alternatively, provided the user decides not to continue the analysis, process 200 continues on to step 270.

[0052] At step 270, the application is executed within

the JVM with the optimized heap parameters. It should be appreciated that step 270 is optional, and that optimized heap parameters can be relied on. However, because the optimized heap parameters are the result of a simulation, it may be desirable to run the application with the optimized heap parameters to ensure their effectiveness.

[0053] Figure 7 is a data flow diagram 700 illustrating data flow between a JVM 705 and a memory usage simulator 715 (e.g., a garbage collection simulator) in accordance with one embodiment of the present invention. It should be appreciated that JVM 705 and memory usage simulator 715 may reside on the same computer system or on different computer systems. The data transferred between JVM 705 and memory usage simulator 715 (e.g., memory usage data 720 and optimized parameters 745) is generally small, and is easy to transfer regardless of where JVM 705 and memory usage simulator 715 reside.

[0054] Java-based application 710 is executed within JVM 705, wherein a memory usage data 720 is generated. Memory usage data 720 is transmitted to memory usage simulator 715. Memory usage data 720 comprises memory usage information including timing information concerning each garbage collection. Memory usage simulator 715 generates a graphical representation 740 of the memory usage data.

[0055] In response to graphical representation 740, at least one parameter 725 is inputted into memory usage simulator 715. Optionally, workload characteristics 730 and simulator configuration 735 are inputted into memory usage simulator 715. Based on memory usage data 720 and input heap parameters 725, a simulation is executed. Specifically, based on memory usage data 720, a number of assumptions are made. For example, a linear rate of allocations between garbage collections is assumed. By assuming the behavior of the garbage collections, a simulation can be performed accounting for changes in the heap parameters. It should be appreciated that other embodiments of the invention could make different assumptions.

[0056] Upon completion of the simulation, in one embodiment, graphical representation 740 is updated to display the results of the simulation. It should be appreciated that the simulation can be repeated any number of times for various heap parameters 725. Upon determination that the heap parameters are optimized, the optimized heap parameters 745 are inserted into the JVM.

[0057] The preferred embodiment of the present invention, a method for optimization of memory usage for a computer application, is thus described. While the present invention has been described in particular embodiments, it should be appreciated that the present invention should not be construed as limited by such embodiments, but rather construed according to the appended claims.

[0058] The disclosures in United States patent application No. 10/104751, from which this application claims priority, and in the abstract accompanying this application are incorporated herein by reference.

Claims

1. A method of optimizing memory usage for a computer application (710), including the steps of:

   a) receiving (215) application memory usage data (720), said application memory usage data (720) comprising timing information;
   b) generating (225) a graphical representation (740) of said application memory usage data (720);
   c) receiving (230) at least one heap parameter (725); and
   d) performing (245) a memory usage simulation based on said application memory usage data (720) and said heap parameter (725).

2. A method as recited in claim 1, wherein said computer program (710) is Java-based.

3. A method as recited in claim 1 or 2, including the step of inserting (260) said heap parameter into a programming language interpreter (705).

4. A method as recited in claim 3, wherein said programming language interpreter (705) is a Java Virtual Machine.

5. A method as recited in any preceding claim, wherein said heap parameter (725) is received in response to a user input.

6. A method as recited in any preceding claim, including the step of repeating steps c) and d) to optimize garbage collection.

7. A method as recited in any preceding claim, including the step of generating (250) a graphical representation (740) of garbage collection simulation.

8. A method as recited in any preceding claim, wherein said memory usage data (720) is received from a memory allocation behaviour trace.

9. A method as recited in claim 8, wherein said memory allocation behaviour trace is recorded as -Xverbosegc data.

10. A method as recited in any preceding claim, wherein step d) includes the steps of:

    receiving (605) said memory usage data;
    determining (610) a simulation model based on

said memory usage data;
determining (615) heap configuration and heap policies based on said memory usage data;
receiving (620) at least one said heap parameter;
determining (625) simulation characteristics for at least one measurement interval of said memory usage data; and
simulating (630) an application run based on said simulation characteristics.

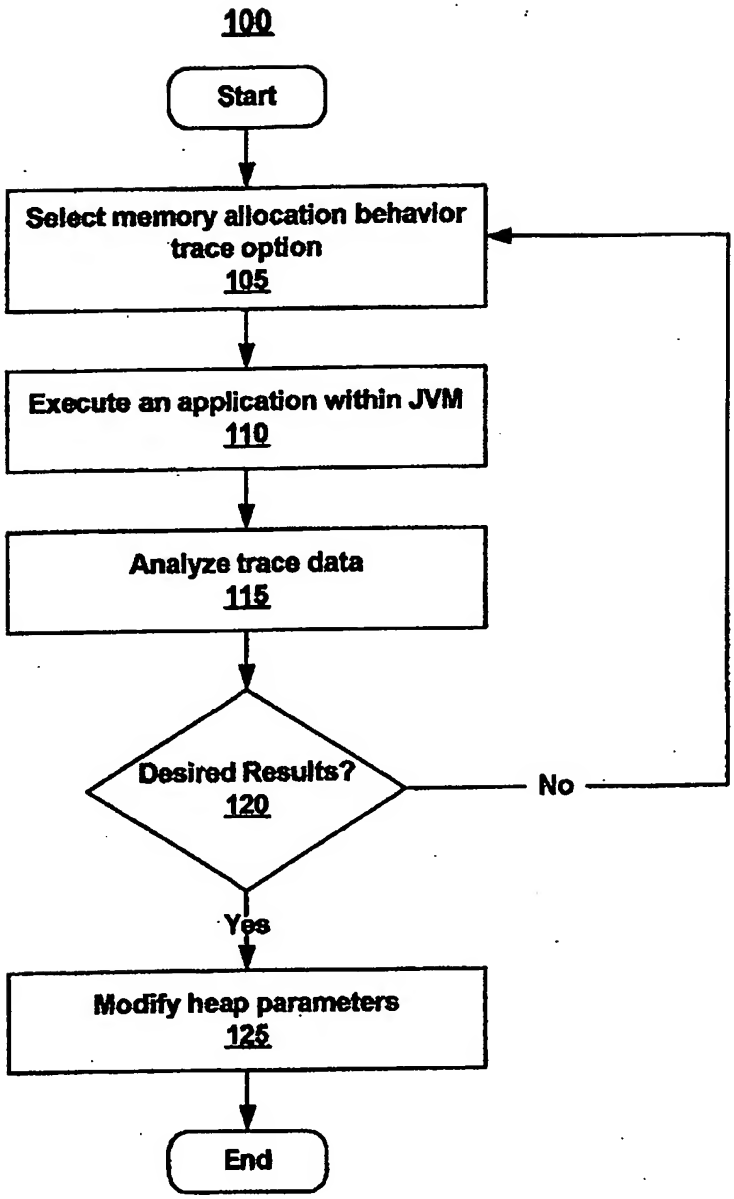


FIG. 1 (Prior Art)

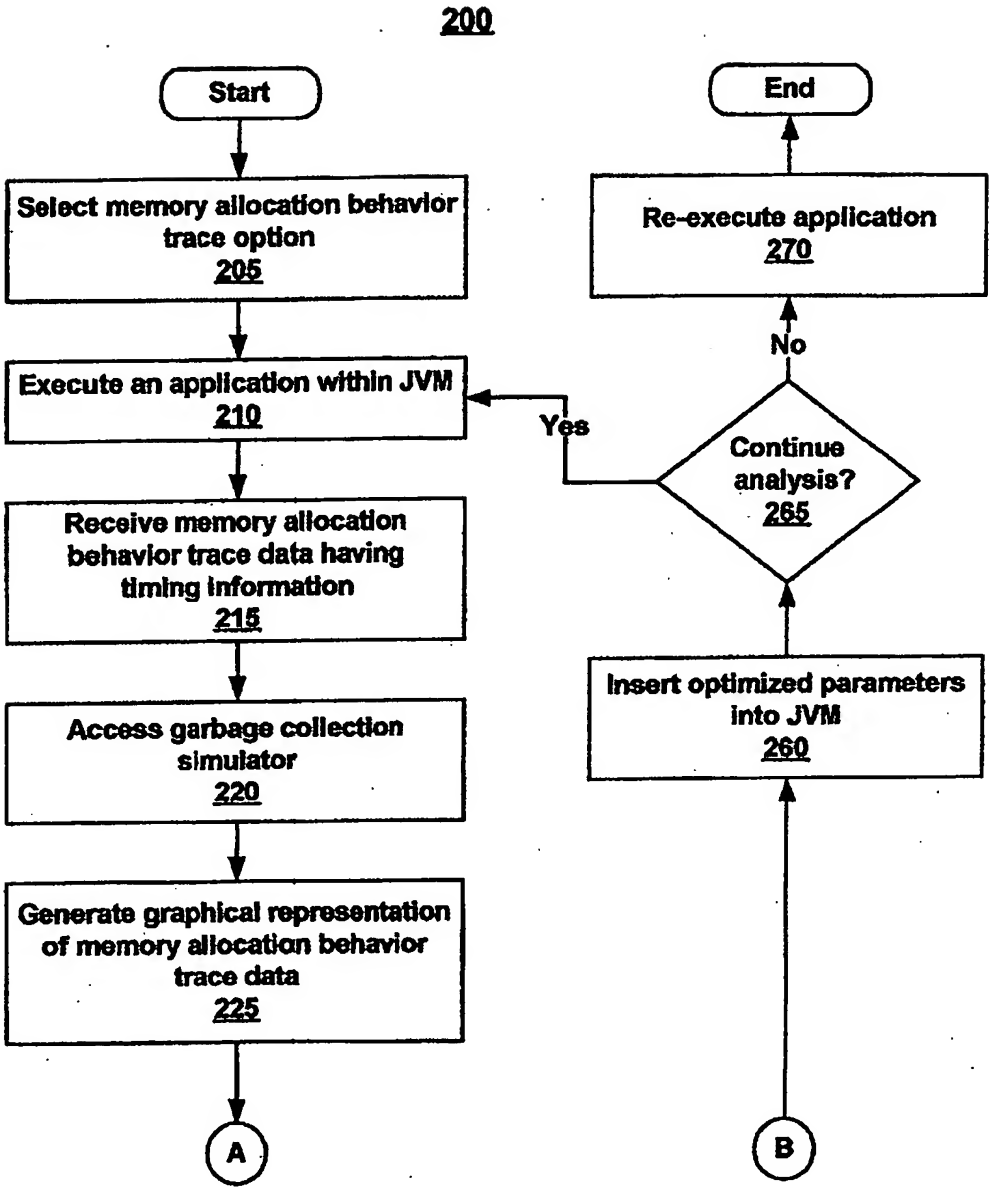

200
Start
Select memory allocation behavior trace option 205
Execute an application within JVM 210
Receive memory allocation behavior trace data having timing information 215
Access garbage collection simulator 220
Generate graphical representation of memory allocation behavior trace data 225
A
End
Re-execute application 270
No
Yes
Continue analysis? 265
Insert optimized parameters into JVM 260
B

FIG. 2A

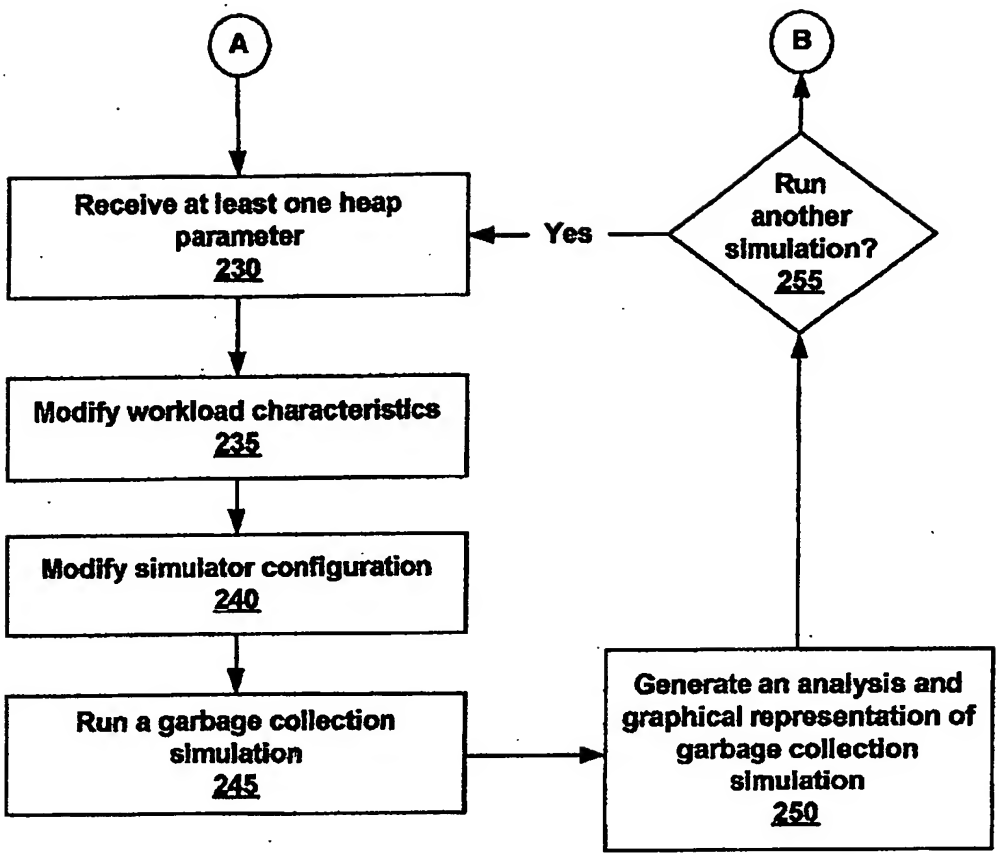

A
B
Receive at least one heap parameter
230
Yes
Run another simulation?
255
Modify workload characteristics
235
Modify simulator configuration
240
Run a garbage collection simulation
245
Generate an analysis and graphical representation of garbage collection simulation
250

FIG. 2B

Figure 3

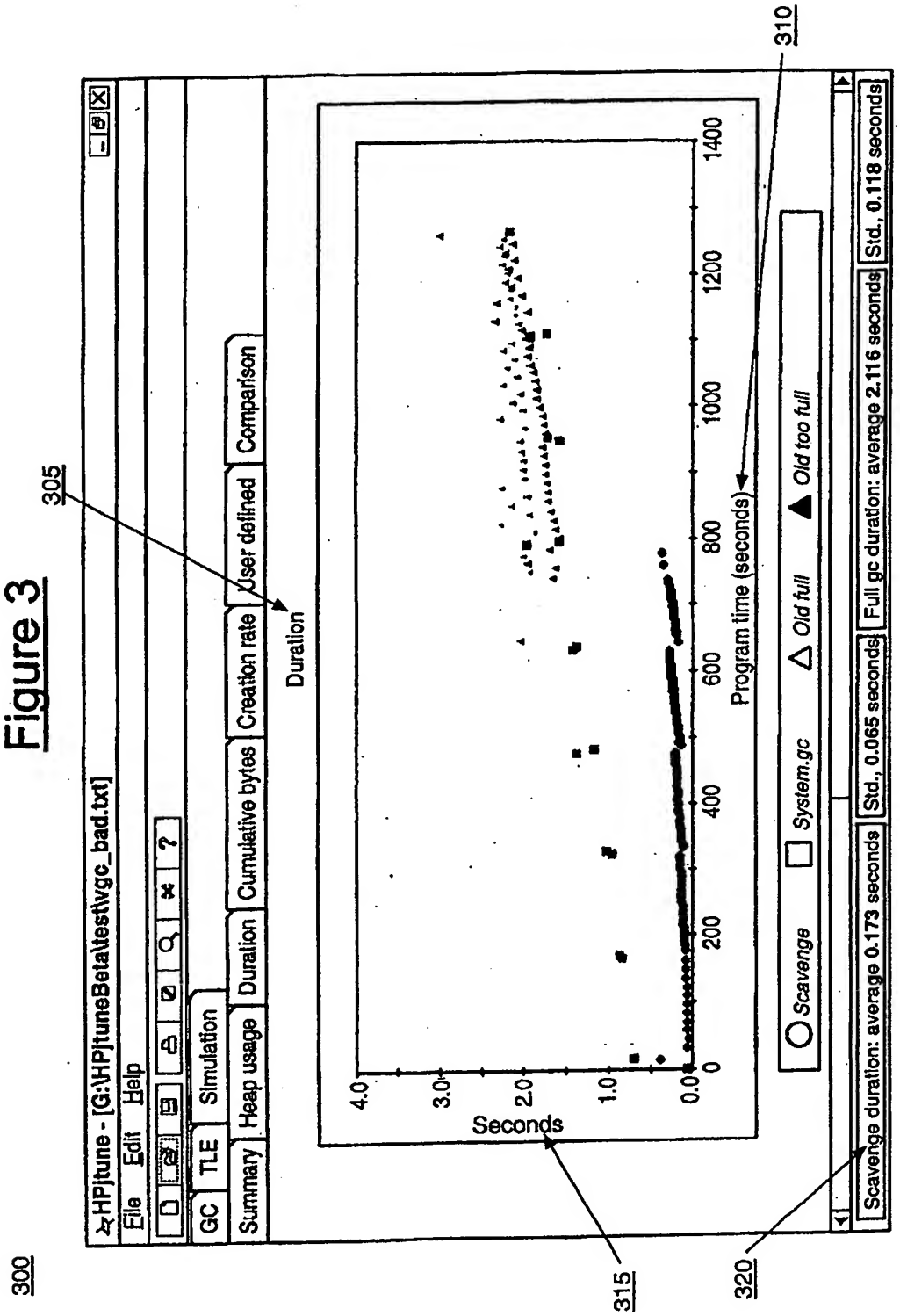

300
305
310
315
320
HPjtune - [G:\HPjtuneBeta\Test\vgc_bad.txt]
File Edit Help
GC TLE Simulation
Summary Heap usage Duration Cumulative bytes Creation rate User defined Comparison
Duration
Seconds
4.0
3.0
2.0
1.0
0.0
0
200
400
600
800
1000
1200
1400
Program time (seconds)
Scavenge
System.gc
Old full
Old too full
Scavenge duration: average 0.173 seconds
Std., 0.065 seconds
Full gc duration: average 2.116 seconds
Std., 0.118 seconds

Figure 4

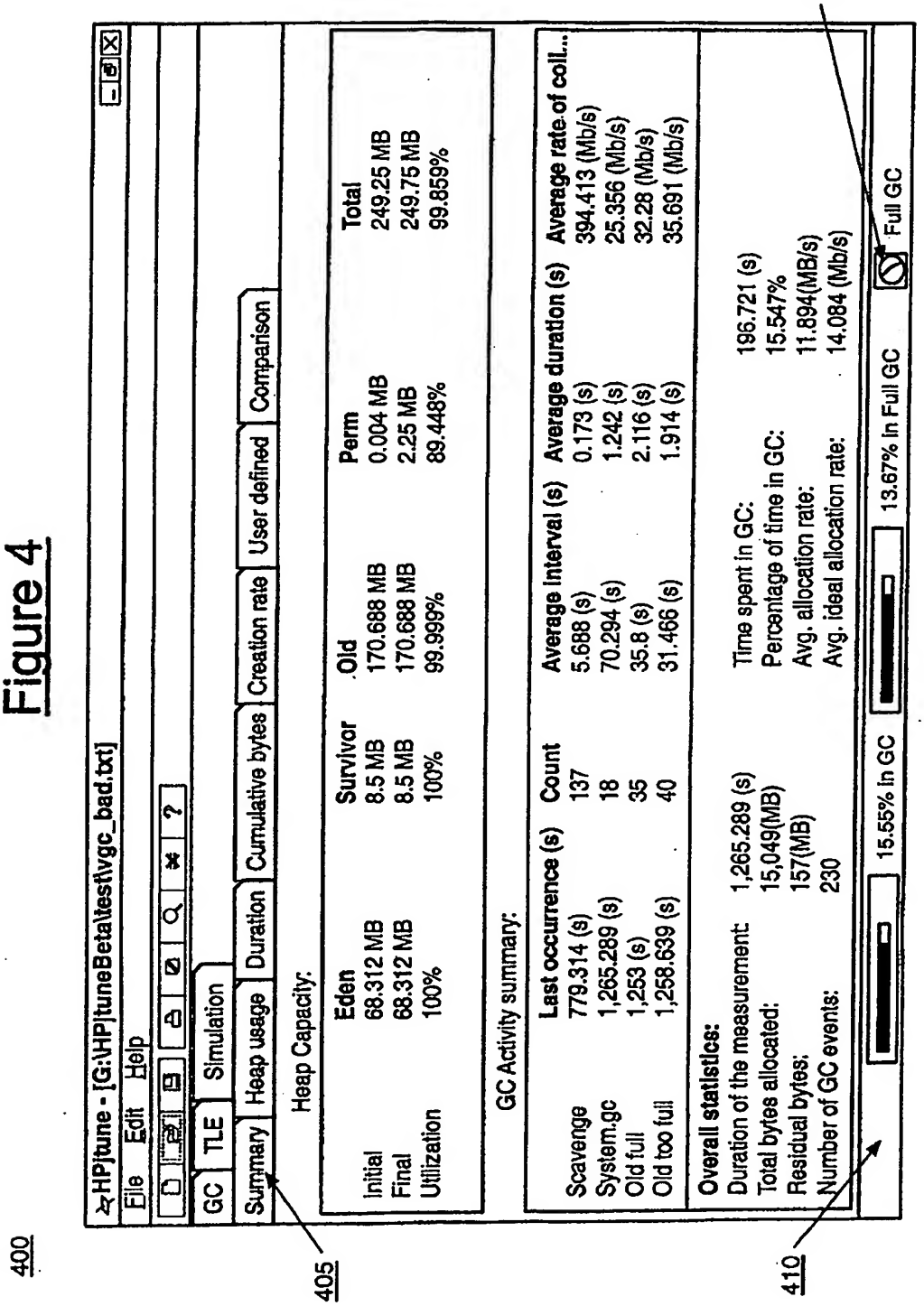
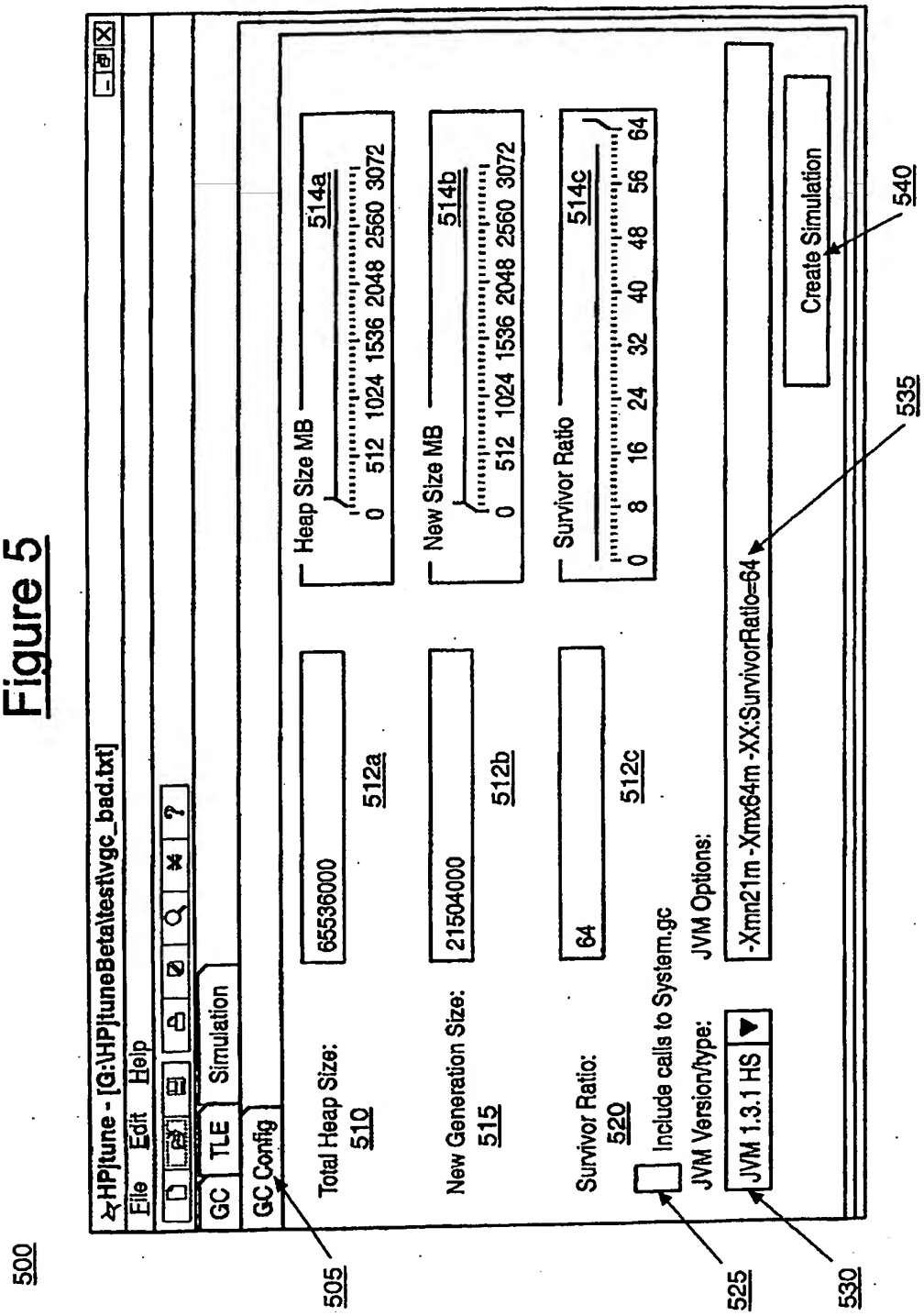

400
HPjtune - [G:\HPjtuneBeta\test\vgc_bad.txt]
File Edit Help
GC TLE Simulation
Summary Heap usage Duration Cumulative bytes Creation rate User defined Comparison
405
Heap Capacity:
Eden Survivor Old Perm Total
Initial 68.312 MB 8.5 MB 170.688 MB 0.004 MB 249.25 MB
Final 68.312 MB 8.5 MB 170.688 MB 2.25 MB 249.75 MB
Utilization 100% 100% 99.999% 89.448% 99.859%
GC Activity summary:
Last occurrence (s) Count Average interval (s) Average duration (s) Average rate of coll...
Scavenge 779.314 (s) 137 5.688 (s) 0.173 (s) 394.413 (Mb/s)
System.gc 1,265.289 (s) 18 70.294 (s) 1.242 (s) 25.356 (Mb/s)
Old full 1,253 (s) 35 35.8 (s) 2.116 (s) 32.28 (Mb/s)
Old too full 1,258.639 (s) 40 31.466 (s) 1.914 (s) 35.691 (Mb/s)
Overall statistics:
Duration of the measurement: 1,265.289 (s)
Total bytes allocated: 15,049(MB)
Residual bytes: 157(MB)
Number of GC events: 230
Time spent in GC: 196.721 (s)
Percentage of time in GC: 15.547%
Avg. allocation rate: 11.894(MB/s)
Avg. ideal allocation rate: 14.084 (Mb/s)
410
15.55% In GC
13.67% In Full GC
Full GC
415

Figure 5


500
HP|tune - [G:\HP|tuneBeta\test\vgc_bad.txt]
File Edit Help
GC TLE Simulation
GC Config
505
Total Heap Size:
510
65536000
512a
Heap Size MB
514a
0 512 1024 1536 2048 2560 3072
New Generation Size:
515
21504000
512b
New Size MB
514b
0 512 1024 1536 2048 2560 3072
Survivor Ratio:
520
64
512c
Survivor Ratio
514c
0 8 16 24 32 40 48 56 64
Include calls to System.gc
525
JVM Version/type:
JVM 1.3.1 HS
JVM Options:
-Xmn21m -Xmx64m -XX:SurvivorRatio=64
530
535
Create Simulation
540

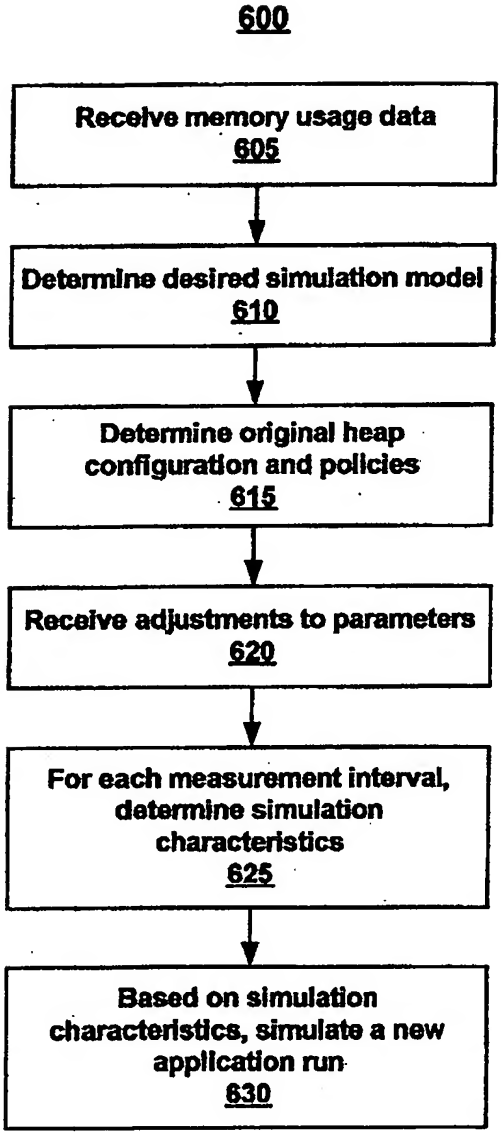

600
Receive memory usage data
605
Determine desired simulation model
610
Determine original heap configuration and policies
615
Receive adjustments to parameters
620
For each measurement interval, determine simulation characteristics
625
Based on simulation characteristics, simulate a new application run
630

FIG. 6

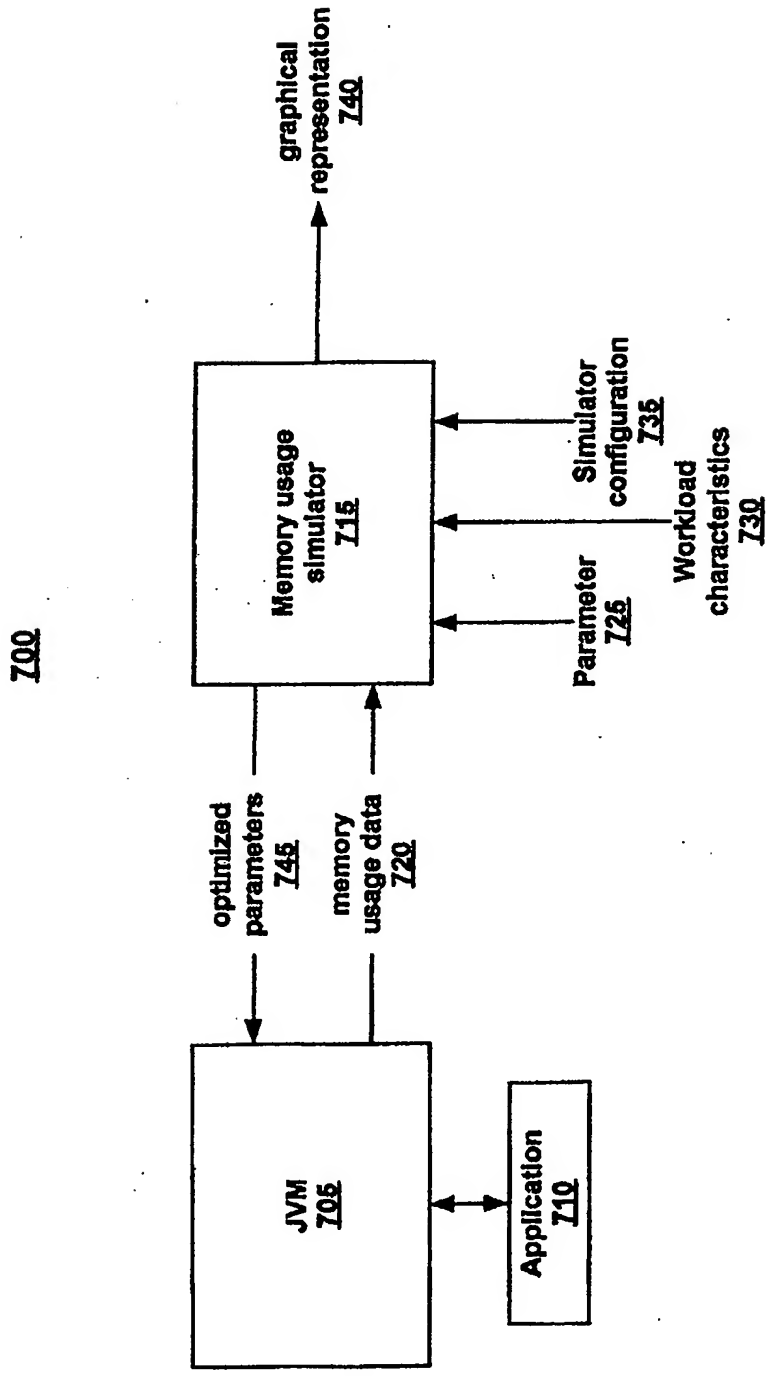

700
graphical representation 740
Memory usage simulator 715
Simulator configuration 735
Workload characteristics 730
Parameter 725
optimized parameters 745
memory usage data 720
JVM 705
Application 710

FIG. 7